Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по работе

Тема: «ARM Учителя»

Выполнил

Студент группы РИС-22-2б

Барыбин К.Ю.

Проверил доц. Кафедры ИТАС

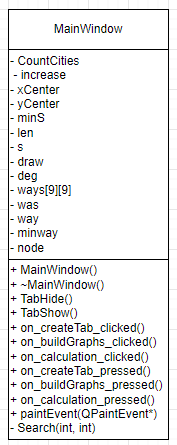
Полякова О. А.

Пермь 2023

**Постановка задачи**

1. Создание и разработка рабочего места учителя.
2. В программе должны генерироваться ученики, выставляться им оценки, быть возможность удалять и добавлять новых учеников. Так же реализовано цветовое разделение оценок.

**Uml-Диаграмма**



**Код программы**

#ifndef STUDENT\_H

#define STUDENT\_H

#include <QString>

*class* **Student** {

*public*:

**Student**(*const* QString& firstName, *const* QString& lastName);

QString **getFirstName**() *const*;

QString **getLastName**() *const*;

int **getGrade**(int index) *const*;

double **getAverageGrade**() *const*;

double **getMathGrade**() *const*;

double **getPhysicsGrade**() *const*;

double **getChemistryGrade**() *const*;

void **setGrade**(int index, int grade);

*private*:

QString m\_firstName;

QString m\_lastName;

int m\_grades[5];

double m\_mathGrade;

double m\_physicsGrade;

double m\_chemistryGrade;

};

#endif *//* *STUDENT\_H*

#include "student.h"

#include <QRandomGenerator>

Student::**Student**(*const* QString& firstName, *const* QString& lastName) :

m\_firstName(firstName),

m\_lastName(lastName),

m\_grades{0, 0, 0, 0, 0},

m\_mathGrade(qrand() % 4 + 2),

m\_physicsGrade(qrand() % 4 + 2),

m\_chemistryGrade(qrand() % 4 + 2)

{

*//* *Случайно* *генерируем* *оценки* *для* *каждого* *ученика*

QRandomGenerator\* generator = QRandomGenerator::global();

*for* (int i = 0; i < 5; i++) {

m\_grades[i] = generator->bounded(2, 6);

}

}

QString Student::**getFirstName**() *const* {

*return* m\_firstName;

}

QString Student::**getLastName**() *const* {

*return* m\_lastName;

}

int Student::**getGrade**(int index) *const* {

*return* m\_grades[index];

}

double Student::**getAverageGrade**() *const* {

int sum = 0;

int count = 0;

*for* (int i = 0; i < 5; i++) {

*if* (m\_grades[i] > 0) {

sum += m\_grades[i];

count++;

}

}

*if* (count > 0) {

*return* *static\_cast*<double>(sum) / count;

} *else* {

*return* 0.0;

}

}

double Student::**getMathGrade**() *const* {

*return* m\_mathGrade;

}

double Student::**getPhysicsGrade**() *const* {

*return* m\_physicsGrade;

}

double Student::**getChemistryGrade**() *const* {

*return* m\_chemistryGrade;

}

void Student::**setGrade**(int index, int grade) {

m\_grades[index] = grade;

}

#ifndef STUDENTMODEL\_H

#define STUDENTMODEL\_H

#include <QAbstractTableModel>

#include <QList>

#include "student.h"

*class* **StudentModel** : *public* QAbstractTableModel {

*public*:

**StudentModel**(QObject\* parent = *nullptr*);

int ***rowCount***(*const* QModelIndex& parent = QModelIndex()) *const* *override*;

int ***columnCount***(*const* QModelIndex& parent = QModelIndex()) *const* *override*;

QVariant ***data***(*const* QModelIndex& index, int role) *const* *override*;

QVariant ***headerData***(int section, Qt::Orientation orientation, int role = Qt::*DisplayRole*) *const* *override*;

bool ***setData***(*const* QModelIndex& index, *const* QVariant& value, int role = Qt::*EditRole*) *override*;

Qt::ItemFlags ***flags***(*const* QModelIndex& index) *const* *override*;

void **addStudent**(*const* QString& firstName, *const* QString& lastName);

void **addStudentManually**();

bool ***insertRows***(int row, int count, *const* QModelIndex& parent = QModelIndex()) *override*;

bool ***removeRows***(int row, int count, *const* QModelIndex& parent = QModelIndex()) *override*;

void **removeStudent**(int row);

*private*:

QList<Student> m\_students;

};

#endif *//* *STUDENTMODEL\_H*

#include "studentmodel.h"

#include <QRandomGenerator>

#include <QInputDialog>

#include <QTableView>

StudentModel::**StudentModel**(QObject\* parent) : QAbstractTableModel(*parent*) {}

int StudentModel::***rowCount***(*const* QModelIndex& parent) *const* {

Q\_UNUSED(parent)

*return* m\_students.size();

}

int StudentModel::***columnCount***(*const* QModelIndex& parent) *const* {

Q\_UNUSED(parent)

*return* 11;

}

QVariant StudentModel::***data***(*const* QModelIndex& index, int role) *const* {

*if* (!index.isValid()) {

*return* QVariant();

}

*if* (role == Qt::*DisplayRole* || role == Qt::*EditRole*) {

*if* (index.column() == 0) {

*return* m\_students[index.row()].getFirstName();

} *else* *if* (index.column() == 1) {

*return* m\_students[index.row()].getLastName();

} *else* *if* (index.column() == 2) {

*return* m\_students[index.row()].getAverageGrade();

} *else* *if* (index.column() >= 3 && index.column() <= 7) {

*return* m\_students[index.row()].getGrade(index.column() - 3);

} *else* *if* (index.column() == 8) {

*return* m\_students[index.row()].getMathGrade();

} *else* *if* (index.column() == 9) {

*return* m\_students[index.row()].getPhysicsGrade();

} *else* *if* (index.column() == 10) {

*return* m\_students[index.row()].getChemistryGrade();

}

} *else* *if* (role == Qt::*BackgroundColorRole*) {

*//* *Устанавливаем* *цвет* *фона* *ячейки* *в* *зависимости* *от* *оценки*

*if* (index.column() >= 3 && index.column() < 8) {

int grade = m\_students[index.row()].getGrade(index.column() - 3);

*if* (grade == 5) {

*return* QColor(Qt::*green*); *//* *Зеленый* *цвет* *для* *оценки* *5*

} *else* *if* (grade == 4) {

*return* QColor(255, 165, 0); *//* *Оранжевый* *цвет* *для* *оценки* *4*

} *else* *if* (grade == 3) {

*return* QColor(Qt::*yellow*); *//* *Желтый* *цвет* *для* *оценки* *3*

} *else* {

*return* QColor(Qt::*red*); *//* *Красный* *цвет* *для* *оценки* *2*

}

}

}

*return* QVariant();

}

QVariant StudentModel::***headerData***(int section, Qt::Orientation orientation, int role) *const* {

*if* (role != Qt::*DisplayRole*) {

*return* QVariant();

}

*if* (orientation == Qt::*Horizontal*) {

*switch* (section) {

*case* 0:

*return* tr("Имя");

*case* 1:

*return* tr("Фамилия");

*case* 2:

*return* tr("Средний балл");

*case* 3:

*return* tr("Оценка 1");

*case* 4:

*return* tr("Оценка 2");

*case* 5:

*return* tr("Оценка 3");

*case* 6:

*return* tr("Оценка 4");

*case* 7:

*return* tr("Оценка 5");

*case* 8:

*return* tr("Ср/б по математике");

*case* 9:

*return* tr("Ср/б по физике");

*case* 10:

*return* tr("Ср/б по химии");

}

}

*return* QVariant();

}

bool StudentModel::***setData***(*const* QModelIndex& index, *const* QVariant& value, int role) {

*if* (index.isValid() && (role == Qt::*EditRole* || role == Qt::*DisplayRole*)) {

*if* (index.column() >= 3 && index.column() < 8) {

int grade = value.toInt();

*if* (grade >= 2 && grade <= 5) {

m\_students[index.row()].setGrade(index.column() - 3, grade);

emit dataChanged(index, index);

*//* *Обновление* *данных* *для* *отображения* *среднего* *балла* *после* *изменения* *оценки*

QModelIndex averageGradeIndex = createIndex(index.row(), 2);

emit dataChanged(averageGradeIndex, averageGradeIndex);

*return* *true*;

}

}

}

*return* *false*;

}

Qt::ItemFlags StudentModel::***flags***(*const* QModelIndex& index) *const* {

*if* (!index.isValid()) {

*return* Qt::*ItemIsEnabled*;

}

*if* (index.column() >= 3 && index.column() < 8) {

*return* QAbstractTableModel::flags(index) | Qt::*ItemIsEditable*;

}

*return* QAbstractTableModel::flags(index);

}

void StudentModel::**addStudent**(*const* QString& firstName, *const* QString& lastName) {

beginInsertRows(QModelIndex(), m\_students.count(), m\_students.count());

m\_students.append(Student(firstName, lastName));

endInsertRows();

}

void StudentModel::**addStudentManually**() {

QString firstName = QInputDialog::getText(*nullptr*, tr("Введите имя"), tr("Имя:"));

QString lastName = QInputDialog::getText(*nullptr*, tr("Введите фамилию"), tr("Фамилия:"));

*if* (!firstName.isEmpty() && !lastName.isEmpty()) {

addStudent(firstName, lastName);

}

}

bool StudentModel::***insertRows***(int row, int count, *const* QModelIndex& parent) {

beginInsertRows(parent, row, row + count - 1);

*for* (int i = 0; i < count; ++i) {

m\_students.insert(row, Student("Имя", "Фамилия"));

}

endInsertRows();

*return* *true*;

}

bool StudentModel::***removeRows***(int row, int count, *const* QModelIndex& parent) {

beginRemoveRows(parent, row, row + count - 1);

*for* (int i = 0; i < count; ++i) {

m\_students.removeAt(row);

}

endRemoveRows();

*return* *true*;

}

void StudentModel::**removeStudent**(int row) {

beginRemoveRows(QModelIndex(), row, row);

m\_students.removeAt(row);

endRemoveRows();

}

#include <QApplication>

#include <QTableView>

#include <QPushButton>

#include <QHBoxLayout>

#include <QListWidget>

#include <QVBoxLayout>

#include <QWidget>

#include <QRandomGenerator>

#include <QInputDialog>

#include "studentmodel.h"

int main(int argc, char\* argv[]) {

QApplication app(*argc*, *argv*);

*//* *Создаем* *модель* *студентов*

StudentModel model;

*//* *Добавляем* *нескольких* *случайно* *сгенерированных* *учеников*

QStringList firstNames = { "Павел", "Кирилл", "Егор", "Василий", "Евгений" };

QStringList lastNames = { "Петров", "Трефелов", "Галкин", "Смирнов", "Иванов" };

*for* (int i = 0; i < 10; ++i) {

QString firstName = firstNames.at(QRandomGenerator::global()->bounded(firstNames.size()));

QString lastName = lastNames.at(QRandomGenerator::global()->bounded(lastNames.size()));

model.addStudent(firstName, lastName);

}

*//* *Создаем* *таблицу* *и* *устанавливаем* *модель*

QTableView tableView;

tableView.*setModel*(*&model*);

*//* *Создаем* *кнопки* *для* *добавления* *и* *удаления* *учеников*

QPushButton addButton("Добавить ученика");

QPushButton removeButton("Удалить ученика");

*//* *Создаем* *горизонтальный* *слой* *для* *кнопок*

QHBoxLayout buttonLayout;

buttonLayout.addWidget(*&addButton*);

buttonLayout.addWidget(*&removeButton*);

*//* *Создаем* *вертикальный* *слой* *для* *таблицы* *и* *кнопок*

QVBoxLayout mainLayout;

mainLayout.addWidget(*&tableView*);

mainLayout.addLayout(*&buttonLayout*);

*//* *Создаем* *виджет* *и* *устанавливаем* *наш* *вертикальный* *слой*

QWidget window;

window.setLayout(*&mainLayout*);

*//* *Подключаем* *слоты* *для* *добавления* *и* *удаления* *учеников*

QObject::connect(&addButton, &QPushButton::clicked, [&](){

QString firstName = QInputDialog::getText(*&window*, "Имя", "Введите имя:");

QString lastName = QInputDialog::getText(*&window*, "Фамилия", "Введите фамилию:");

model.addStudent(firstName, lastName);

});

QObject::connect(&removeButton, &QPushButton::clicked, [&](){

QModelIndexList selectedIndexes = tableView.selectionModel()->selectedRows();

*if* (selectedIndexes.size() > 0) {

int row = selectedIndexes[0].row();

model.removeStudent(row);

}

});

window.show();

*return* app.exec();

}

**Текст выступления**

Здравствуйте, меня зовут Барыбин Кирилл, я являюсь студентом ПНИПУ группы РИС-22-2Б. Я хочу вам представить Творческую работу, состоящую из двух частей. Первая часть – это автоматизированное рабочее место учителя, а вторая – решение задачи коммивояжёра. Обе части реализованы на языке С++ и визуально представлены в Qt.

Итак, рассмотрим ARM учителя. Программа реализуют управление группой учеников, выставление их оценок, а также добавление и удаление учеников. Эти функции реализованы, с помощью двух классов student и studentModel, рассмотрим каждый из них по отдельности. **(Student.h)** Класс Student представляет модель студента, в которой хранятся данные о каждом студенте. В заголовочном файле хранится объявление функций, которые используются в **Student.cpp**

* Функция Student::Student(const QString& firstName, const QString& lastName) является конструктором класса Student. Она принимает две строки: firstName (имя студента) и lastName (фамилия студента). Конструктор инициализирует соответствующие члены данных класса, включая имя, фамилию и оценки студента. Оценки генерируются случайным образом с использованием генератора случайных чисел QRandomGenerator.
* Функция QString Student::getFirstName() const является методом для получения имени студента. Она возвращает значение члена данных m\_firstName, содержащего имя студента.
* Функция QString Student::getLastName() const является методом для получения фамилии студента. Она возвращает значение члена данных m\_lastName, содержащего фамилию студента.
* Функция int Student::getGrade(int index) const является методом для получения оценки студента по указанному индексу. Она принимает индекс в качестве параметра и возвращает значение из соответствующего элемента массива m\_grades.
* Функция double Student::getAverageGrade() const является методом для получения среднего балла студента. Она вычисляет среднее значение оценок, хранящихся в массиве m\_grades. Если оценок нет, то есть все элементы массива равны 0, функция возвращает 0.0.
* Функции double Student::getMathGrade() const, double Student::getPhysicsGrade() const и double Student::getChemistryGrade() const являются методами доступа для получения оценок студента по математике, физике и химии соответственно. Они возвращают значения соответствующих членов данных (m\_mathGrade, m\_physicsGrade и m\_chemistryGrade).
* Функция void Student::setGrade(int index, int grade) является методом установки оценки студента по указанному индексу. Она принимает индекс и оценку в качестве параметров и устанавливает значение в соответствующий элемент массива m\_grades.

Дальше аналогично рассмотрим класс StudentModel, он представляет модель данных для работы со списком студентов. В заголовочном файле StudentModel.h хранится объявление функций и методов используемых в StudentModel.cpp

* StudentModel(QObject\* parent) - конструктор класса StudentModel, инициализирует родительский объект и создает пустую модель данных.
* int rowCount(const QModelIndex& parent) const - возвращает количество строк в модели данных. В данном случае, это количество студентов в списке m\_students.
* int columnCount(const QModelIndex& parent) const - возвращает количество столбцов в модели данных. В данном случае, это 11 столбцов, представляющих различные атрибуты студента.
* QVariant data(const QModelIndex& index, int role) const - возвращает данные для указанного индекса и роли. В данном случае, он возвращает соответствующие данные для отображения в ячейке таблицы.
* QVariant headerData(int section, Qt::Orientation orientation, int role) const - возвращает данные для заголовка таблицы. В данном случае, он возвращает текстовую метку для указанного раздела и ориентации.
* bool setData(const QModelIndex& index, const QVariant& value, int role) - устанавливает данные для указанного индекса, он используется для изменения оценки студента.
* Qt::ItemFlags flags(const QModelIndex& index) const - возвращает флаги для указанного индекса. Он определяет, какие операции разрешены для конкретной ячейки таблицы.
* void addStudent(const QString& firstName, const QString& lastName) - добавляет нового студента в список m\_students на основе переданных имени и фамилии.
* void addStudentManually() - добавляет нового студента в список m\_students, запрашивая ввод данных у пользователя.
* bool insertRows(int row, int count, const QModelIndex& parent) - вставляет указанное количество строк в модель данных, начиная с указанного ряда.
* bool removeRows(int row, int count, const QModelIndex& parent) - удаляет указанное количество строк из модели данных, начиная с указанного ряда.
* void removeStudent(int row) - удаляет студента из списка m\_students по указанному индексу строки.

Теперь перейдем к основной функции int main, давайте рассмотрим какие действия в ней выполняются:

1. Создается объект приложения QApplication, который представляет графическое приложение Qt.
2. Создается модель StudentModel, которая используется для хранения данных о студентах и их оценках.
3. В цикле for создаются случайно сгенерированные ученики с помощью класса Student и добавляются в модель StudentModel.
4. Создается объект QTableView, который представляет таблицу для отображения данных.
5. Устанавливается модель StudentModel для таблицы QTableView.
6. Создаются кнопки QPushButton для добавления и удаления учеников.
7. Создаются горизонтальный и вертикальный слои (QHBoxLayout и QVBoxLayout) для расположения элементов интерфейса.
8. Создается объект QWidget и устанавливается вертикальный слой в качестве макета виджета.
9. Подключаются слоты (функции-обработчики событий) для кнопок добавления и удаления учеников.
10. Отображается главное окно приложения с помощью метода show().
11. Запускается главный цикл обработки событий приложения с помощью метода app.exec(), который ожидает событий и вызывает соответствующие слоты.

Таким образом, программа создает графический интерфейс с таблицей, в которой отображаются данные о студентах (имя, фамилия, оценки и средний балл), а также позволяет добавлять и удалять учеников с помощью кнопок.

Теперь запустим программу и рассмотрим ее визуальную часть. При запуске программы открывается окно с таблицей случайно сгенерированных учеников, также им выставлены случайные оценки и посчитан средний балл, еще есть средние баллы по математике, физике, химии, чтобы учитель мог видеть среднюю успеваемость ученика. Оценки по предмету выделяются цветом, 2-красный, 3-желтый, 4-оранжевый, 5-зеленый. Эти оценки мы можем менять, средний цвет тоже будет изменяться и средний балл будет пересчитываться. В нижней части окна программы есть две кнопки для добавления и удаления учеников, при нажатии на кнопку добавить ученика будет открываться диалоговое окно, где нужно ввести имя и фамилию ученика. Для удаления учеников нужно выделить строку или же строки таблицы и нажать кнопку удалить ученика и они будут по очереди удаляться.

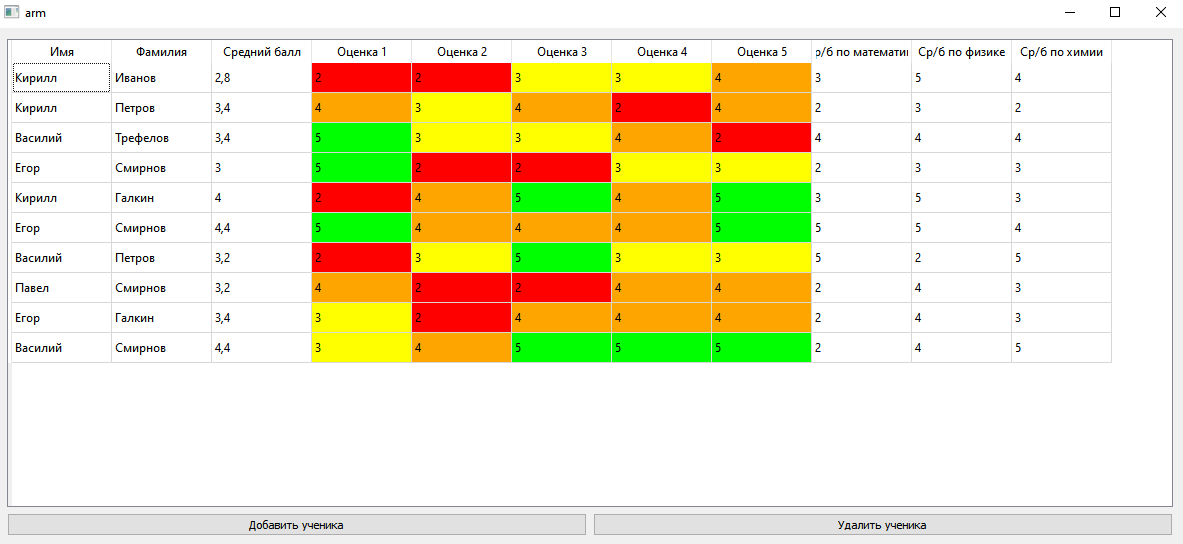
Перейдем ко второй части творческой работы, а именно к решению задачи коммивояжёра. Цель задачи: пройти все пункты только один раз, при этом маршрут должен быть оптимальным. Итак, рассмотрим код в заголовочном файле GraphWidget.h определяются две структуры данных: Point (те пункт) и Edge (те ребро). Структура Point представляет собой точку с координатами x и y. Структура Edge представляет собой ребро между двумя точками, содержит индексы исходной и целевой точек, а также вес ребра. Затем перечислены публичные методы класса, более подробно рассмотрим их реализацию в файле GraphWidget.cpp. В приватных членах класса определены указатели, переменные, векторы, а так же два метода updateMinDistance() - обновляет значение переменной minDistance, вызывает метод solveTSP() для решения задачи коммивояжера и выводит информацию о минимальном расстоянии, calculateDistance() - вычисляет расстояние между двумя точками.

Рассмотрим файл GraphWidget.cpp В этом файле определены следующие функции и методы:

* GraphWidget(QVector<Point>& points, QVector<Edge>& edges, QWidget\* parent) - конструктор класса, который инициализирует виджет с использованием родительского класса QGraphicsView. В конструкторе создается объект QGraphicsScene для отображения элементов графа, устанавливается размер сцены и устанавливается заголовок окна. Также создается поле для вывода текста textOutput и добавляются кнопки для добавления/удаления точек и ребер. Затем вызывается метод solveTSP() для решения задачи коммивояжера и вывода информации о минимальном расстоянии на textOutput.
* drawPoints() - метод для отображения точек графа. Для каждой точки создается объект QGraphicsEllipseItem, представляющий круглую метку, и объект QGraphicsTextItem, представляющий номер точки. Оба элемента добавляются на сцену.
* drawLines() – метод, аналогичный drawPoints(), он используется для отображения ребер графа. Для каждого ребра создается объект QGraphicsLineItem, представляющий линию между двумя точками, и объект QGraphicsTextItem, представляющий вес ребра. Оба элемента добавляются на сцену.
* solveTSP() - метод для решения задачи коммивояжера. В этом методе используется алгоритм полного перебора всех возможных путей между точками графа. Для каждого пути вычисляется его длина, а затем выбирается путь с минимальной длиной. Минимальное расстояние и оптимальный путь сохраняются в соответствующих переменных.
* drawOptimalPath() - метод для отображения оптимального пути на сцене. Он использует оптимальный путь, сохраненный в переменной optimalPath, для создания объекта QGraphicsLineItem для каждого ребра пути. Ребра отображаются синим цветом.
* addPoint() - метод для добавления новой точки в граф. Пользователю предлагается ввести координаты новой точки, которая затем добавляется в вектор points. Также вызываются методы solveTSP() и updateMinDistance() для обновления решения задачи коммивояжера и вывода нового минимального расстояния.
* removePoint() - метод для удаления точки из графа. Пользователю предлагается ввести индекс точки, которую нужно удалить. Точка удаляется из вектора points, а затем вызываются методы solveTSP() и updateMinDistance().
* addEdge() - метод для добавления нового ребра в граф. Пользователю предлагается ввести индексы и вес ребра. Ребро добавляется в вектор edges, а также отображается на сцене с помощью объекта QGraphicsLineItem и QGraphicsTextItem. Затем вызываются методы solveTSP() и updateMinDistance().
* removeEdge() - метод для удаления ребра из графа. Пользователю предлагается ввести индексы исходной и конечной точек ребра, которое нужно удалить. Ребро удаляется из вектора edges, а также сцена очищается и заново отрисовываются точки и оставшиеся ребра. Затем вызываются методы solveTSP() и updateMinDistance().

Запустим программу и протестируем ее работу. При запуске открывается окно в котором отрисован граф, 4 кнопки, и поле для вывода минимальной длинны пути. Удалим ребра 2-6 и 5-6, и удалим сам пункт 6, длина пути автоматически пересчиталась. Теперь добавим новый пункт, ему автоматически выдадут номер, по координатам, например -100, 350. Длинна ребра рассчитывается автоматически, по длине вектора. Добавим векторы из пункта 2 и 5, после внесения этих изменений длина пути коммивояжёра обновилась, а так же его путь.

**Вывод**

****